Utilizing the Untapped Potential of Indirect Encoding for Neural Networks with Meta Learning

Adam Katona[1], Nuno Lourenço[2], Penousal Machado[2],
Daniel W. Franks[1], and James Alfred Walker[1]

[1] Department of Computer Science, University of York, UK
{ak1774,daniel.franks,james.walker}@york.ac.uk
[2] CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
{naml,machado}@dei.uc.pt

Abstract. Indirect encoding is a promising area of research in machine learning/evolutionary computation, however, it is rarely able to achieve performance on par with state of the art directly encoded methods. One of the most important properties of indirect encoding is the ability to control exploration during learning by transforming random genotypic variation into an arbitrary distribution of phenotypic variation. This gives indirect encoding a capacity to learn to be adaptable in a way which is not possible for direct encoding. However, during normal objective based learning, there is no direct selection for adaptability, which results in not only a missed opportunity to improve the ability to learn, but often degrading it too. The recent meta learning algorithm MAML makes it possible to directly and efficiently optimize for the ability to adapt. This paper demonstrates that even when indirect encoding can be detrimental to performance in the case of normal learning, when selecting for the ability to adapt, indirect encoding can outperform direct encoding in a fair comparison. The indirect encoding technique Hypernetwork was used on the task of few shot image classification on the Omniglot dataset. The results show the importance of directly optimizing for adaptability in realizing the powerful potential of indirect encoding.

Keywords: Indirect encoding · Evolvability · Meta learning · Neuroevolution · Hypernetwork · HyperNEAT

1 Introduction

Most deep learning research is done with the natural representation of neural networks, where each weight in the network directly maps on to a separate parameter in the representation. We call this a direct encoding. On the other hand, in an indirect encoding the weights do not directly map on to the representation, and instead, we apply a transformation to the representation to produce the weights. In Evolutionary Computation (EC), this transformation is commonly referred to as the genotype-phenotype mapping.

Direct encoding seems to work well and we can successfully train models with as many as 175 billion parameters [4]. As such, direct encoding dominates practically all benchmark problems. Natural evolution on the other hand uses indirect encoding. It is debatable whether evolution is as successful a problem solver because it uses indirect encoding or despite it. There could be many reasons why nature ended up with indirect encoding, be it biological limitations or because indirect encoding provides benefits. However, when designing our learning algorithms, we are faced with the decision of using either. This raises the question: Is there any advantage in using an indirect encoding?

1.1 The Potential of Indirect Encoding

In this work we argue that indirect encoding is worthy of our attention because it has two interesting properties, which direct encoding lacks:

1. Indirect encoding can control the exploration during learning by making changes in promising directions more sensitive and changes in less promising directions insensitive.
2. Indirect encoding can reuse parameters multiple times, making it possible to learn regular structures.

Controlling Exploration Indirect encoding is capable of controlling exploration during learning by modifying the type of variation mutations can cause. This is possible since the genotype-phenotype map has the ability to transform random genotypic variation to an advantageous distribution of phenotypic variation [31].

A simple example which is often used to demonstrate this property is how nature encodes development plans for symmetric bodies [17,19]. Because of the way the developmental program for the body is encoded, it is easier for evolution to change the length of both limbs together, then to change them separately, which is probably a useful way to explore possible space of body configurations.

A similar concept that describes the same phenomena is developmental canalization. Indirect encoding has the ability to entrench certain phenotypic features, making them difficult to change, the same way water can dig a canal, making the path of future flow more stable. A great example of how canalization can emerge in artificial evolution is given in [17]. In their experiments, developmental canalization made certain good quality variation more common, increasing the ability to innovate.

One of the most successful algorithms that controls its own exploration is Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [14]. CMA-ES uses an exploration strategy of adapting the covariance matrix of a multivariate Gaussian distribution in a way that more promising directions are explored more. By using indirect encoding we can allow the algorithm to automatically discover exploration strategies instead of manually inventing and incorporating them into our algorithms.

Reusing Parameters Another important property of indirect encoding is the ability to reuse parameters multiple times. We argue that this is actually a similar property to controlling exploration, since having many separate parameters that always change together, or having one parameter which is reused many times achieves similar results.

Indirect encoding allows the reuse of information to build regular and modular structures [16,25]. We already know how beneficial some structures are, for example the convolution is reusing the convolutional kernel weights many times at different locations in the image, which makes learning vision tasks efficient. By using indirect encoding we open the possibility to discover such useful structures automatically, even when using a fully connected architecture [8].

The Vision of Indirect Encoding By using an indirect encoding, we can allow our algorithms to automatically discover different modular architectures, and exploration strategies, instead of manually inventing and coding them. The vision of automatically discovering these kinds of representations with the ability to learn effectively for many kinds of problems is extremely alluring [25].

Richard Sutton's bitter lesson argument [30] postulates that general methods which scale well with computation eventually always outperform methods for which the AI researchers build in extra handcrafted knowledge. For this reason, researchers should concentrate their main efforts on studying general and scalable methods. We hypothesise that utilising indirect encoding will allow us to discover efficient learning systems automatically in a data-driven way in the spirit of Sutton's argument.

1.2 The Difficulties with Indirect Encoding

As we discussed in the previous section, indirect encodings have a powerful capability to control their own exploration during training. Since the ability to improve further has no effect on current fitness, greedy algorithms are not expected to select for representations which result in good exploration strategies. Because there are many more bad exploration strategies than good ones, if there is no selection for the ability to adapt, the exploration strategy will just drift, causing indirect encoding to most likely hurt learning performance. This leads to the main hypothesis of this paper.

Hypothesis Greedy learning algorithms are unlikely to make full use of the capabilities of indirect encoding.

We do not suggest however that it is impossible to make use of indirect encoding capabilities without selecting for the ability to adapt, only that it is much more difficult. Several researchers in the field of evolution of evolvability [1,31,20] argue that evolvability can emerge without selection in an unsupervised way. Huizinga et al. [17] showed that developmental canalization can emerge in a divergent search like environment.

Much effort was given to algorithms which instead of selecting for individuals with the ability to improve their fitness, select for the ability to generate diverse behaviour in their offspring [19,10]. These algorithms capture a different aspect of evolvability which might be able to utilize the capabilities of indirect encoding just as well.

In the rest of the section, we evaluate our hypothesis in the context of past results with indirect encoding.

HyperNEAT HyperNEAT [26] is one of the most well known indirect encoding techniques for neuroevolution. There are several demonstrations of how HyperNEAT outperforms direct encoding in different domains [7,3,11], especially if the task is more regular [6]. These results seemingly contradict our hypothesis, since HyperNEAT does not directly select for the ability to adapt. We argue however, that this is not the case for three reasons.

First, HyperNEAT uses innovation protection. Innovation protection was originally introduced in NEAT [27], and it keeps innovative genes in the gene pool even if they have a poor performance. The justification for it is that when we change the topology, fitness likely decreases first until the weights of the new structural elements can be fine-tuned. However, innovation protection also helps to protect individuals that are better at evolving, since it provides them with a few generations to prove their ability to improve. This means that HyperNEAT is actually indirectly selecting for the ability to adapt. More experiments are necessary to evaluate whether HyperNEAT would perform well without innovation protection, but this is beyond the scope of this paper.

Second, the baselines for these results were using some version of NEAT such as plain NEAT, Fixed Topology NEAT (FT-NEAT) or Perceptron NEAT (P-NEAT). While NEAT might be a good algorithm to evolve the small query networks for HyperNEAT, it might not be an ideal baseline for the larger directly encoded networks. NEAT changes parameters one by one, recent results suggest that techniques which modify many weights at the same time perform much better for large networks, like CMA-ES [15], GA [29] and ES [22]. When we compare the performance of HyperNEAT to these more efficient baselines, on the task of learning to play Atari games, direct encoding seems to have the advantage in most games [22].

Finally, HyperNEAT experiments are typically using relatively small networks. Choromanska et al. [5] showed that as the network size increases, the number of bad quality local optima are diminishing exponentially. The problem of local optima, which is a very important factor in the case of small networks, is less important for large networks. For this reason, the results obtained in many HyperNEAT experiments are not necessarily expected to generalize to large scale networks.

Differentiable Pattern Producing Network One of the most impressive achievements of indirect encoding is the demonstrated ability to invent convolution from scratch using a fully connected architecture [8]. The DPPN (Dif-

ferentiable Pattern Producing Network) is a differentiable version of the CPPN (Compositional Pattern Producing Network) [24] used in HyperNEAT. In this work, the authors run experiments with three different settings. In the Darwinian setting, individuals were evaluated on their ability to solve the task without further adaptation. In the case of the Baldwinian setting, individuals were allowed to learn further by using gradient descent, resulting in selection for the ability to adapt. In the case of the Lamarckian setting, the situation is the same as with Baldwinian evolution with the additional feature of inheriting the learned weights as well. In the case of the Darwinian setting, without selection for the ability to adapt, the task was not solved successfully, not a single digit was recognisable in the image reconstruction task. When they used Baldwinian or Lamarckian evolution however performance was much better, and convolution-like fully connected weights were generated. This experiment supports our hypothesis that selection for the ability to learn is crucial in realizing the full capabilities of indirect encoding.

Hypernetworks A recent indirect encoding technique called Hypernetworks [13] can achieve near state of the art performance on sequence modelling tasks. This result was achieved with dynamic Hypernetworks, where a recurrent network is enhanced with the new ability to modify its own weights during inference, based on the current input and state of the network. The simpler static Hypernetwork does not change the capabilities of the network, only the way the parameters are represented, making the comparison between direct and indirect encoding easy. When using static Hypernetworks to generate the weights of a convolutional network, the results on an image classification task are worse than direct encoding. This result however was obtained by using around an order of magnitude less parameter for the indirect encoding. We show in section 3.4. that we achieve similar results when using Hypernetworks with the same number of parameters, indirect encoding cannot outperform direct encoding on an image classification task with greedy learning.

2 Background

In this paper we combine ideas from the fields of Deep Learning and Evolutionary Computation, so we use the terminology from both fields to describe similar concepts. For example, we use the terms evolvability and the ability to adapt which are similar concepts, both are concerned with potential yet unrealized improvements, but imply a different underlying algorithm. The situation is the same with the phrases “selecting for” or “optimizing for”.

2.1 Indirect Encoding for Neuroevolution

There is a vast literature covering indirect network encoding. The field of artificial embriogeny is concerned with these techniques, a great review is available by [28]. Relatively few of these techniques were constructed with modern deep

learning scale in mind (millions or billions of connections). In this section, we discuss two families of techniques, which were shown to be viable for these large networks.

One family of methods to indirectly encode the weights of a neural network is to use query function and a substrate [26]. The query function is a parameterized function; typically a small neural network [24], which maps the coordinates of a source and a target neuron to a single weight between the source and target neuron. These coordinates for the neurons come from the substrate, which is often manually crafted by placing each neuron in 2D or 3D space, or it can also be learned [21]. A conceptual diagram of how this kind of encoding can represent weights can be seen in Fig. 1.

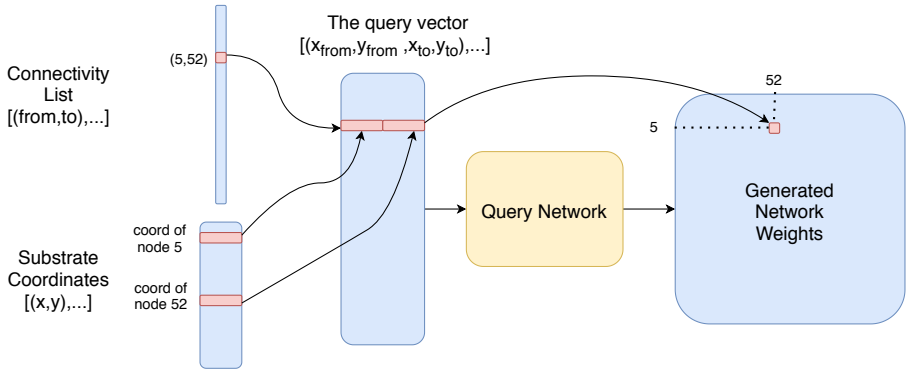

Fig. 1. Query networks: In the case of query networks, each node in the neural network is assigned a coordinate in space, which is called the substrate (this example shows a 2D space with coordinates x and y). For each connection, a query vector is assembled from the coordinates of the source and target neurons. The weight for each connection is determined by evaluating the query vector with the query network, which is a small neural network. To generate the whole network as many forward passes are necessary as there are connections in the network. Yellow blocks represent the learned parameters, blue blocks represent fixed or generated values, and the red blocks show an example of how a single weight is generated

To calculate the weights of the whole network, we need to query this network as many times as many weights there is. For a large network, this could mean millions of queries, which could become prohibitively expensive. This is especially problematic in cases when we only use the network a few times before updating, like supervised learning, and less problematic in control or reinforcement learning problems, where we use the network hundreds or thousands of times before updating it. This is only an issue during training since during inference time the network does not change anymore. Luckily the size of the query network is typically very small, and due to the large number of queries, they can effectively utilize the GPU.

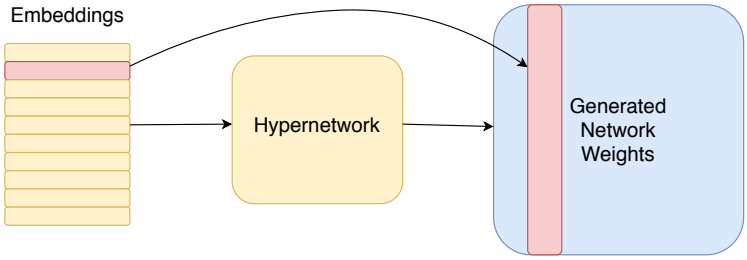

Fig. 2. Hypernetworks: Each embedding is transformed into a chunk of weights by the Hypernetwork [13]. Yellow blocks represent the learned parameters, the blue block represents the generated weights, and the red blocks show an example of how a single embedding is transformed into a chunk of generated weights. See Fig. 4 on how a Hypernetwork can be used as part of a larger model

Another family of methods is to simply transform an embedding space to the weight space. The first method to use this technique for large neural networks is Hypernetworks [13] which uses a simple linear projection as transformation. A conceptual diagram of how this kind of encoding can represent weights can be seen in Fig. 2. The learned parameters are a set of embeddings and the parameters of the projection network. Each embedding is then used to generate a separate part of the network. This separation into smaller chunks is required to keep the size of the transformation manageable. Another side effect of this separation, which motivated the invention of the technique is that there is a kind of information sharing between these chunks, since they are projected from the same subspace.

2.2 MAML

In this work, we use the meta learning algorithm MAML [9] as an algorithm which can optimize for the ability to adapt. The goal of MAML is to find initial parameters that allow for fast adaptation for many different tasks. It consists of 2 different kinds of updates. There is a so-called fine-tuning step, which is a normal gradient step that changes the parameters θ so the training task $\mathcal{D}_i^{tr}$ loss is lower, as seen in equation 1. Then there is the meta update, which updates the initial or meta parameters, so fine-tuning can achieve good generalization performance on the test tasks $\mathcal{D}_i^{ts}$. Calculating the meta gradient requires us to differentiate through a gradient step (see equation 2), which means we also need to calculate second order gradients.

$$\theta' = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{tr}) \tag{1}$$

$$\mathcal{L}_{meta} = \sum_{task\ i} \mathcal{L}(\theta', \mathcal{D}_i^{ts}) = \sum_{task\ i} \mathcal{L}(\theta - \alpha \nabla_\theta L(\theta, \mathcal{D}_i^{tr}), \mathcal{D}_i^{ts}) \tag{2}$$

In this work, we used the gradient based version of MAML because both our models and the task are differentiable. The evolutionary version of the algorithm

ES MAML [23] can be used in cases when either the model or the task or both are not differentiable. This property of ES MAML might be interesting for research into indirect encoding since there are many exotic and interesting nondifferentiable ways to represent networks [28].

3 Experiment

The goal of our experiments is to evaluate our original hypothesis, that indirect encoding is unlikely to be beneficial in case of greedy learning, but can lead to better performance when the ability to adapt is selected.

We used two kinds of vision tasks, simple image classification on the FashionMNIST [32] dataset for greedy learning and few shot classification on the Omniglot [18] dataset for meta learning. The problem setting of few shot image classification is shown in Fig. 3

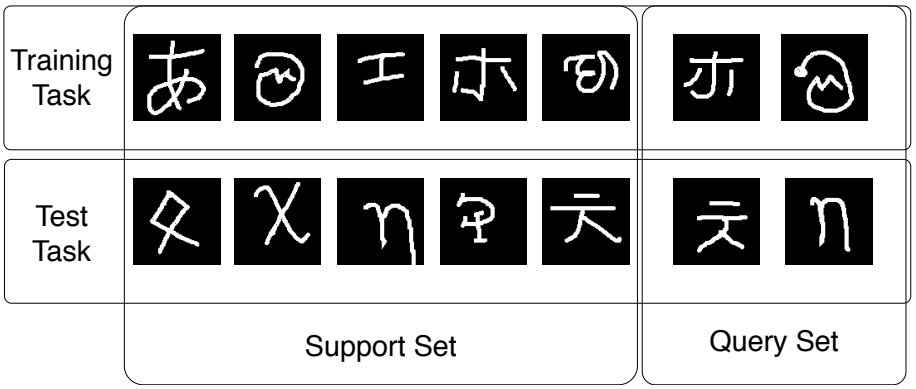

Fig. 3. Few shot learning problem. The goal of MAML is to find model parameters that can be fine-tuned given the few examples in the support set (in this example 5 way 1 shot, there are 5 different classes with 1 example from each), so they can accurately classify the images in the query set. The training tasks created by randomly sampling 5 out of the 1200 training classes. The performance is evaluated on the test tasks, which are created by sampling 5 out of the 400 test classes.

We used fully connected networks because convolutional networks are already very good at vision tasks and we wanted to leave room for improvement. Because the two dataset uses the same resolution 28 by 28 we could use the same networks without any modification for both tasks.

3.1 Fair Comparison

To determine whether indirect encoding is beneficial for learning, we need to establish a baseline with direct encoding. To make the comparison fair, we used

approximately the same number of parameters to encode the exact same networks with both direct and indirect encoding. We used 4 different sized networks, which are summed up in Table 1.

Table 1. Dimensions of the networks, and the number of parameters used in both direct and indirect encoding. The table also shows the hyperparameters used for indirect encoding.

	Hidden dims	Direct parameters	Indirect parameters	$[z_{dim1}, N_{in1}, N_{out1}]$	$[z_{dim2}, N_{in2}, N_{out2}]$
Tiny	[32,16]	25,829	25,977	[14,2,2]	[16,2,2]
Small	[64,32]	52,677	51,237	[14,4,4]	[16,2,2]
Medium	[128,64]	109,445	107,229	[30,4,4]	[16,2,2]
Large	[256,128,64,64]	247,621	244,837	[56,4,4]	[32,4,4]

For indirect encoding, we generated the weights of the first two hidden layers (the vast majority of parameters), the biases and the rest of the weights were encoded directly, as shown in Fig. 4. The authors of the original Hypernetwork paper used a single Hypernetwork to generate the weights of all layers. They argue that this constrained the system to share some commonality between the layers, which resulted in decreased performance [13]. This would especially be the case for fully connected networks since the intermediate fully connected representations lack the common structure that the intermediate representations of convolutional networks have. For this reason, we used separate Hypernetworks for the two generated layers.

3.2 Implementation Details

We used the same formulation of the Hypernetwork as in [13], instead of a simple projection, 2 matrix multiplications are used to project the embeddings into weights. First, the embeddings with size z_{dim} are projected into the shape N_{in} by z_{dim}. The second projection then projects the result of the previous projection into the shape $[N_{out},N_{in},unit_{dim}]$. Where $unit_{dim}$ is the size of the smallest chunk of weights generated. N_{out},N_{in} are hyperparameters controlling how much weights should be generated from a simple embedding. Doing the projection this way is equivalent to a simple large projection but uses way fewer parameters because the weights of the second matrix are reused many times. We choose $unit_{dim}$ to be the number of connections a single neuron has in the generated layer.

Initializing weights is an important aspect of training neural networks to avoid the vanishing or exploding gradient problem. Normally we would want to initialize our weight in a way that the magnitude of the activations throughout the network stays constant. This can be achieved by initializing each layer so their gain is one [12]. Normally in a fully connected layer, the variance of the

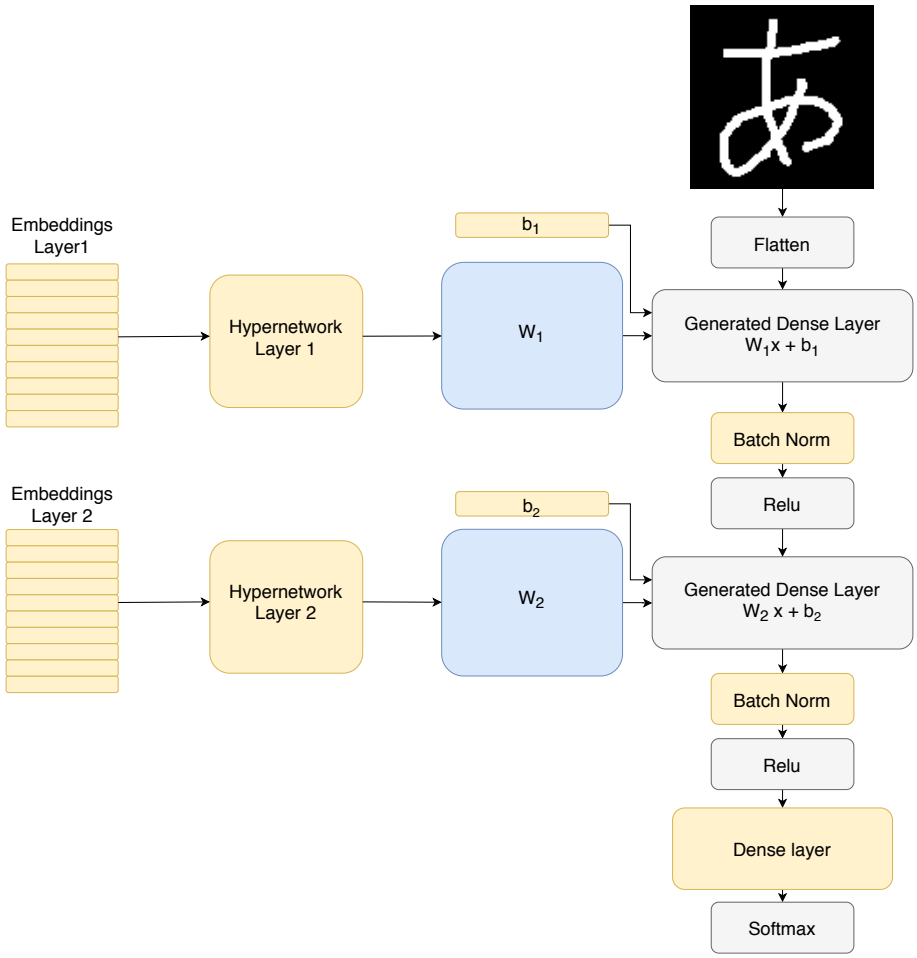

Fig. 4. The indirect architecture used in the experiments. The yellow boxes are learned parameters, the blue boxes are generated parameters and the gray boxes are functions. The direct encoding uses the same architecture, but the two dense layers are represented normally.

activations in a layer depends on the variance of the inputs, the variance of the weights, and the number of neurons in the previous layer as shown in equation 3.

$$Var(a_l) = n_{l-1} * Var(W_l) * Var(a_{l-1}) \tag{3}$$

In the case of Hypernetworks however we use one network to generate the weights of another network. We initialized the Hypernetwork in a way that will result in the generated layer to have a gain of one. In the case of Hypernetworks, because the matrix weights in the second projection are reused multiple times, we need to use the number of neurons contributing to a single weight, which is the number of embeddings, z_{dim} (equation 5). In the following equations, W_1 is the first matrix and W_2 is the second matrix in the Hypernetwork.

$$Var(a_1) = z_{dim} * Var(W_1) * Var(a_0) \tag{4}$$

$$Var(a_2) = z_{dim} * Var(W_2) * Var(a_1) \tag{5}$$

$$Var(a_2) = z_{dim} * Var(W_2) * z_{dim} * Var(W_1) * Var(a_0) \tag{6}$$

Let the gain of the Hypernetwork be equal to the required variance of the generated layer for it to have a gain of one, and provide the additional constraint of $Var(W_1) = Var(W_2)$

$$Var(a_2)/Var(a_0) := 1/n_{generated\ fan\ in} \tag{7}$$

$$Var(W) = \sqrt{\frac{1}{n_{generated\ fan\ in} * z_{dim} * z_{dim}}} \tag{8}$$

The source code for all of our experiments are available at https://github.com/adam-katona/indirect_encoding_maml

3.3 Greedy Learning Experiment

To evaluate the effect of indirect encoding on performance in the case of a greedy learning algorithm, we used the FashionMNIST dataset for image classification. The networks were trained with gradient descent. The batch size was 64, we used the Adam optimizer with a learning rate of 0.001. Each run was repeated 20 times, test results are show in Fig. 5 and in Table 2.

Indirect encoding achieves slightly but consistently worse test accuracies for all network sizes. For all but the tiny network the difference is significant ($p<0.01$, Mann-Whitney U test). These are similar result reported in [13], showing the inability of greedy learning to benefit from the capabilities of indirect encoding, supporting our original hypothesis.

Table 2. Median test accuracies (out of 20 runs) achieved on the FashionMNIST dataset.

	Direct	Indirect
Tiny	**0.8723**	0.8718
Small	**0.8840**	0.8817
Medium	**0.8903**	0.8862
Large	**0.8937**	0.8904

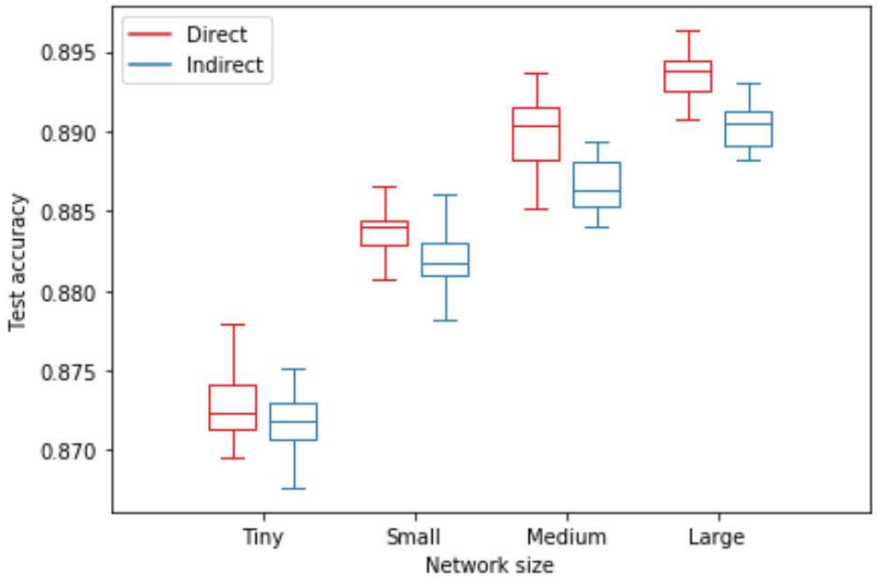

Fig. 5. Greedy learning results: Final test accuracy on the FashionMNIST dataset. Without selecting for adaptability, the direct encoding slightly but consistently outperforms indirect encoding in a fair comparison. The difference is significant for the small,medium and large networks ($p<0.01$, Mann-Whitney U test)

3.4 Meta Learning Experiment

To evaluate the performance of indirect encoding when we are optimizing for the ability to adapt, we run experiments with few shot learning on the Omniglot dataset. We use 5 way 1 shot learning. We followed the procedure described in [9]. We used 1200 characters for training and 400 for testing. We augmented the characters by applying multiples of 90° rotations. We used batch normalization in the same way as in the original MAML implementation, only using batch statistics and not accumulating running statistics. We used a learnable per step, per parameter learning rate for the fine-tuning update, as proposed in [2]. We used cosine annealing meta learning rate schedule, as proposed in [2]. We used a meta batch size of 32. We used the Adam optimizer and the initial meta learning rate of 0.005 using cosine annealing learning rate scheduler with a restart period of 3000 meta batches. We trained each model for 50000 iterations (meta batches). We used a single adaptation step while training, and used three while evaluating performance on the test tasks, the same way as done in the original MAML paper [9].

Each run was repeated 8 times, test results are show in Fig. 6 and in Table 3. For the tiny, small, and medium networks, indirect encoding achieved higher accuracies. For the small and medium sizes the difference is significant ($p<0.01$, Mann-Whitney U test). For the large network, indirect encoding has slightly lower accuracy than direct encoding.

The result with the large network is surprising since for all other network configurations indirect encoding had the advantage. For the large network we added 2 additional directly encoded layers. We suspect that there is some kind of interesting interactions between the direct and indirect layers which hinders performance.

Table 3. Median test accuracies (out of 8 runs) achieved on 5-way, 1-shot learning on the Omniglot dataset.

	Direct	Indirect
Tiny	0.739	**0.754**
Small	0.775	**0.806**
Medium	0.818	**0.839**
Large	**0.875**	0.872

4 Conclusion

We proposed the hypothesis that greedy learning is unlikely to benefit from the capabilities of indirect encoding, and selecting for the ability to adapt is necessary. We verified the previously demonstrated [13] results that the indirect encoding technique Hypernetworks achieves lower accuracy when trained on an

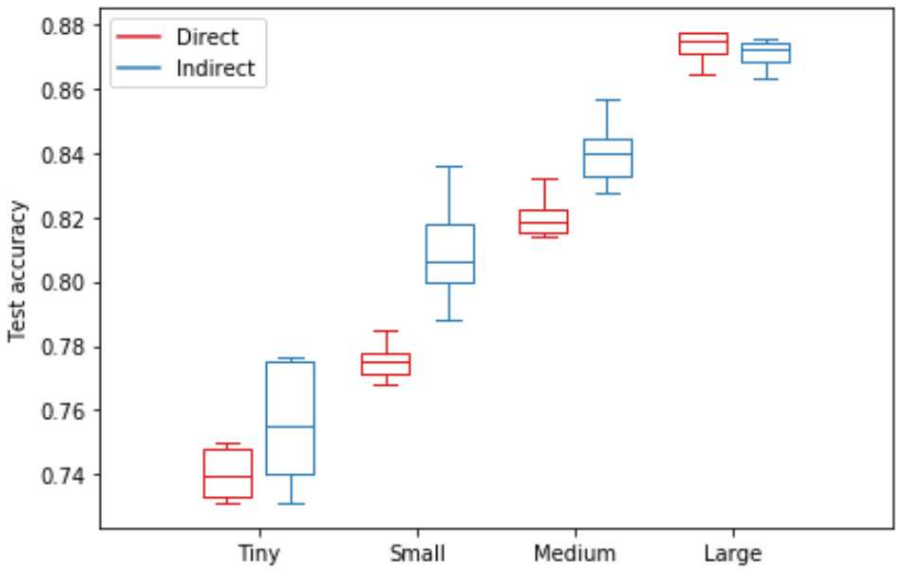

Fig. 6. Meta learning results: Final test accuracies after the third gradient step on the test tasks of 5-shot, 1-way image classification. When adaptability is selected, the indirect encoding outperforms the direct encoding in a fair comparison for most network sizes, except when networks are large.

image classification task compared to direct encoding. We then showed that when the ability to adapt is selected with MAML, Hypernetworks can outperform direct encoding on an image classification task. Our results suggest that optimizing for the ability to adapt is indeed of key importance when learning with indirect encoding. More experiments are needed in different domains to verify whether the hypothesis holds in a more general setting. We hope that our results will motivate other researchers to explore the exciting possibilities of utilizing meta learning to realize the powerful potential of indirect encoding.

Acknowledgement

This work was supported by the EPSRC Centre for Doctoral Training in Intelligent Games & Game Intelligence (IGGI) [EP/L015846/1] and the Digital Creativity Labs funded by EPSRC/AHRC/Innovate UK [EP/M023265/1]. This work was partially supported by Society for the Promotion of Evolutionary Computation in Europe and its Surroundings (SPECIES).

References

1. Altenberg, L., et al.: The evolution of evolvability in genetic programming. Advances in genetic programming **3**, 47–74 (1994)

2. Antoniou, A., Edwards, H., Storkey, A.: How to train your maml. arXiv preprint arXiv:1810.09502 (2018)
3. Assunção, F., Lourenço, N., Machado, P., Ribeiro, B.: Using gp is neat: Evolving compositional pattern production functions. In: European Conference on Genetic Programming. pp. 3–18. Springer (2018)
4. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)
5. Choromanska, A., Henaff, M., Mathieu, M., Arous, G.B., LeCun, Y.: The loss surfaces of multilayer networks. In: Artificial intelligence and statistics. pp. 192–204 (2015)
6. Clune, J., Ofria, C., Pennock, R.T.: How a generative encoding fares as problem-regularity decreases. In: International Conference on Parallel Problem Solving from Nature. pp. 358–367. Springer (2008)
7. Clune, J., Ofria, C., Pennock, R.T.: The sensitivity of hyperneat to different geometric representations of a problem. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. pp. 675–682 (2009)
8. Fernando, C., Banarse, D., Reynolds, M., Besse, F., Pfau, D., Jaderberg, M., Lanctot, M., Wierstra, D.: Convolution by evolution: Differentiable pattern producing networks. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 109–116 (2016)
9. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400 (2017)
10. Gajewski, A., Clune, J., Stanley, K.O., Lehman, J.: Evolvability es: scalable and direct optimization of evolvability. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 107–115 (2019)
11. Gauci, J., Stanley, K.O.: A case study on the critical role of geometric regularity in machine learning. In: AAAI. pp. 628–633 (2008)
12. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256 (2010)
13. Ha, D., Dai, A., Le, Q.V.: Hypernetworks. arXiv preprint arXiv:1609.09106 (2016)
14. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In: Proceedings of IEEE international conference on evolutionary computation. pp. 312–317. IEEE (1996)
15. Hausknecht, M., Lehman, J., Miikkulainen, R., Stone, P.: A neuroevolution approach to general atari game playing. IEEE Transactions on Computational Intelligence and AI in Games **6**(4), 355–366 (2014)
16. Huizinga, J., Clune, J., Mouret, J.B.: Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. pp. 697–704 (2014)
17. Huizinga, J., Stanley, K.O., Clune, J.: The emergence of canalization and evolvability in an open-ended, interactive evolutionary system. Artificial life **24**(3), 157–181 (2018)
18. Lake, B., Salakhutdinov, R., Gross, J., Tenenbaum, J.: One shot learning of simple visual concepts. In: Proceedings of the annual meeting of the cognitive science society. vol. 33 (2011)
19. Mengistu, H., Lehman, J., Clune, J.: Evolvability search: directly selecting for evolvability in order to study and produce it. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 141–148 (2016)

20. Pigliucci, M.: Is evolvability evolvable? Nature Reviews Genetics **9**(1), 75–82 (2008)
21. Risi, S., Stanley, K.O.: Enhancing es-hyperneat to evolve more complex regular neural networks. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation. pp. 1539–1546 (2011)
22. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864 (2017)
23. Song, X., Gao, W., Yang, Y., Choromanski, K., Pacchiano, A., Tang, Y.: Es-maml: Simple hessian-free meta learning. arXiv preprint arXiv:1910.01215 (2019)
24. Stanley, K.O.: Compositional pattern producing networks: A novel abstraction of development. Genetic programming and evolvable machines **8**(2), 131–162 (2007)
25. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. Nature Machine Intelligence **1**(1), 24–35 (2019)
26. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artificial life **15**(2), 185–212 (2009)
27. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)
28. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. Artificial Life **9**(2), 93–130 (2003)
29. Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O., Clune, J.: Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567 (2017)
30. Sutton, R.: The bitter lesson. Incomplete Ideas (blog), March **13**, 12 (2019)
31. Watson, R.A., Szathmáry, E.: How can evolution learn? Trends in ecology & evolution **31**(2), 147–157 (2016)
32. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)